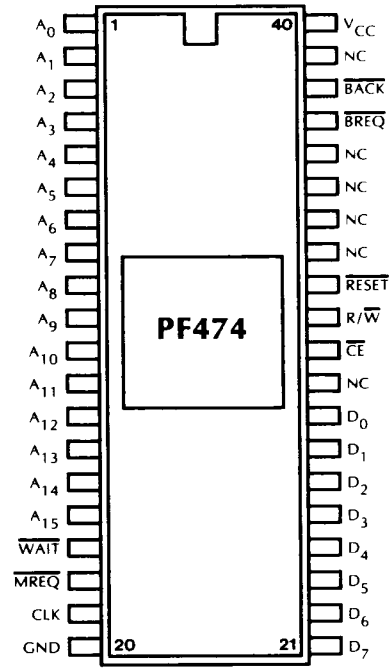
INTRODUCTION

The **PF474** microcircuit is a totally new device which implements the revolutionary String Proximity Computer and Ranker on a single VLSI silicon chip using high-speed NMOS technology. The String Proximity Computer (United States and foreign patents pending) is a state-of-the-art device that compares two symbol strings to arrive at a numeric rating of their similarity; a 32-bit binary fraction ranging between zero and one. This highly sophisticated measure of similarity is very flexible and adaptable to a wide spectrum of applications. It is useful to think of the PF474 as a scoring device that produces high scores for very similar strings and low scores for highly dissimilar strings. Typically, the PF474 is used to search a list of symbol strings (a database for example) for entries that are most similar to a query which is fixed for the duration of the operation. When the operation is complete, the ranker portion of the device can be accessed to identify which members of the list received the highest scores.



FEATURES

- Computes 32-bit Proximity Values
- Processes strings with lengths of up to 127 8-bit symbols
- Maximum throughput of 66,110 comparisons and rankings per second for strings of 8 characters (PF474-40)
- Parameter tables, stored in externally accessible RAM, allow tailoring of the Proximity Function
- Maintains internal 16-element ranked list of best matches
- Permits location of 15 additional *next best* matches
- Modern programming architecture allows the device to be accessed as normal memory
- High-speed DMA facility permits rapid loading from external memory: up to 2 million bytes per second
- Smart DMA permits optional editing of the input data to offload certain useful preprocessing tasks from the host processor
- Fits naturally and simply into microprocessor based systems
- LS TTL compatible
- Range of clock speeds:
  - 2.5 MHz for PF474-25
  - 3.0 MHz for PF474-30
  - 3.6 MHz for PF474-36
  - 4.0 MHz for PF474-40
- High-reliability 40-pin ceramic DIP package
- Single +5 volt supply

The Challenge: Finding Approximately Matching Strings

The retrieval of information based on inexact, incomplete or inaccurate data has been a difficult problem for computers to cope with. Many tasks such as looking up a name, correcting spelling errors or searching a natural language database, demand that the computer be able to perform *pattern matching* on strings, that is, to be able to recognize and retrieve strings which are similar to or approximately equal to a query string.

In the past, systems which have attacked this kind of problem in real time have typically used special purpose algorithms which are limited in their capabilities. Most schemes for the automatic correction of spelling errors require that the misspelling be incorrect by no more than three deletions, insertions and substitutions of letters. These systems are software implementations which are constrained by the requirement to rapidly locate a few words within a database of many tens of thousands of words.

Generally, such a system can be conceptually broken down into two parts: first, a method of limiting the search space to a small number of strings and, second, a string matching function which identifies the most similar records in the restricted search space. The search space is restricted because the string matching algorithms have only been able to make up to a few thousand comparisons per second. Unfortunately, restricting the search space may reduce the effectiveness of the system by decreasing the robustness of the search. Hence, the system may work fine with differences of up to a certain degree, but beyond that, the system may totally fail. For instance, the above described spelling correction system would be completely unable to recognize *youthinasia* as being similar to *euthanasia*, or *shartroose* as close to *chartreuse*.

The Solution: The PF474 Microcircuit

Now, there is a general purpose solution to the problem of approximate string matching: a powerful string comparison computer which operates at extremely high speeds. The PF474, a VLSI integrated circuit developed by Proximity Technology Inc., computes a powerful string comparison function at the high speed of 20,000 to 30,000 comparisons per second and faster. Thus many search problems may be solved by exhaustive brute force methods within an acceptable amount of time.

The Proximity Function, as implemented in the PF474, is very flexible and highly adaptable to specific problems. This is important, for example, because an optimal comparison method for correcting the spelling of English words will not work quite as well for comparing proper names, French words, and so forth.

In most applications, one string is chosen as a search string or *query string*. This query string is compared with each string in a large database by the PF474, which computes a Proximity Value for each of the database strings. The highest Proximity Values (corresponding to the best matches) are stored in a ranked list in the PF474 for reference at the end of the search.

The Proximity Function is unique among string matching algorithms in that it can be computed quickly in hardware. Furthermore, the comparison time is *linear* with the number of characters in the strings. Thus, for strings of length 8, the comparison rate is 66,110 comparisons per second, and for strings of length 127, the comparison rate is 5,164 per second. (These speed figures assume a system providing strings to a 4 MHz PF474 at its full input rate.) This is quite remarkable since other similar functions generally require a computation time proportional to the *square* of the length of the strings. A full mathematical description of the actual algorithm can be found in appendix A.

1.2 PF474 ARCHITECTURAL OVERVIEW

The overall architecture and data flow of the PF474 is shown in the diagram below.

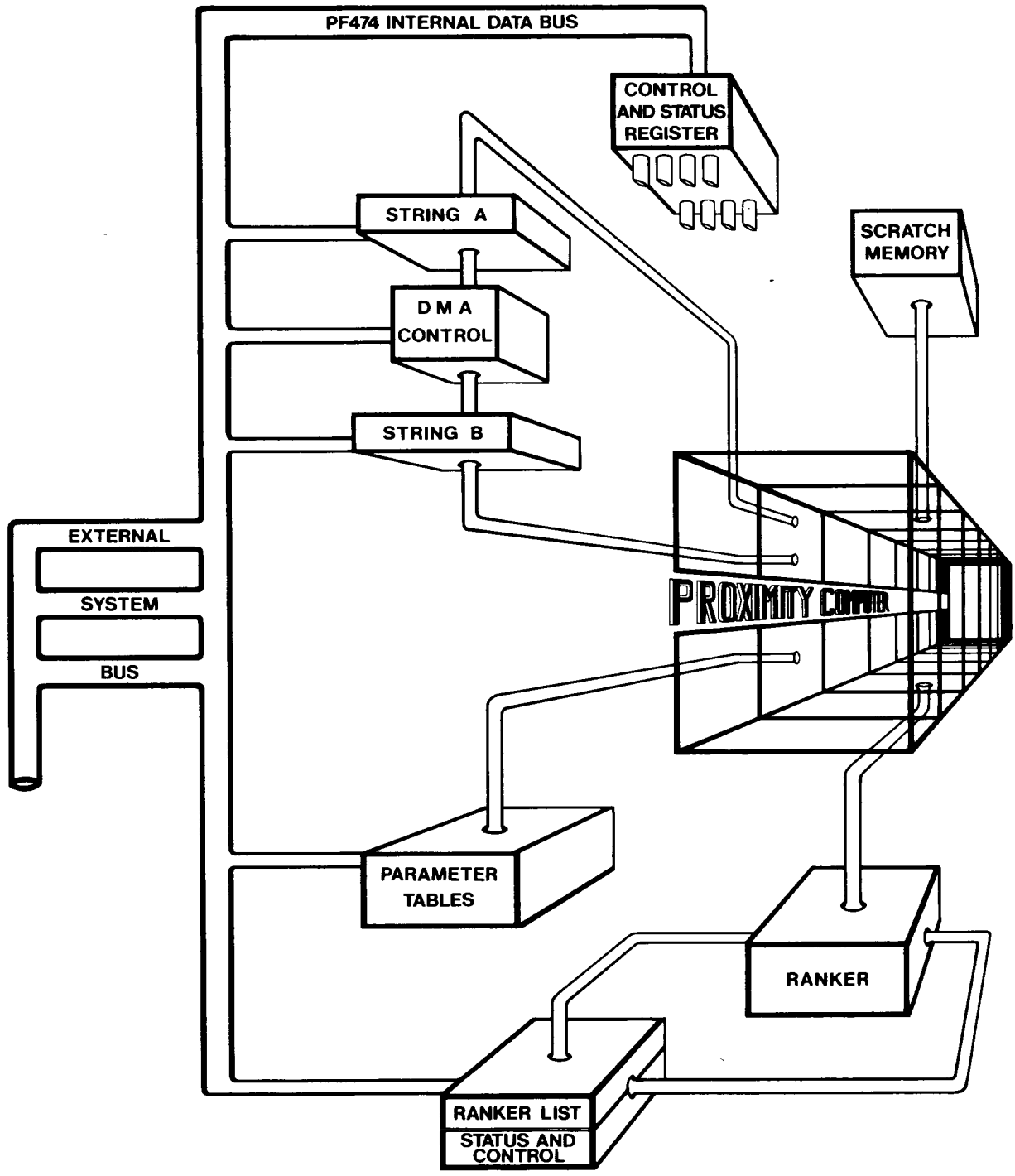


Figure 1. Architectural Blocks

System access to the PF474 is through the External System Bus. The four memory regions that are externally accessible are the Control and Status Registers, Strings A and B, the Parameter Table, and the Ranker Control and its Status Registers and Ranked List. The string memories are loaded directly under CPU control, or by Direct Memory Access data transfer under control of the DMA controller. When the proper commands are written to the Command Register, the Proximity Computer performs a string comparison operation on the two strings in String A and B. During the calculations, the Scratch Memory is used for intermediate results, and the Parameter Table is consulted. The results of the Proximity Computation are then sent to the Ranker. The Status Register now indicates that the Proximity Computer is ready to start another computation, but that the Ranker is still busy. The Ranker updates the Ranked List with the result of the latest comparison. The size of the Ranked List and other details of the Ranker's operation are controlled by the Ranker Control Registers.

The three computational subsystems – DMA Controller, Proximity Computer, and Ranker – merit further discussion.

1.2.1 DMA Capabilities

The DMA (Direct Memory Access) Controller of the PF474 supports high–speed data loading in DMA mode. This computational subsystem of the PF474 controls the source, destination, and mode of editing for high–performance DMA input to the PF474.

To initiate DMA mode, software writes a starting address into a register inside the PF474. The PF474 then requests control of the system bus and reads system memory sequentially until the NULL character is encountered. This terminates DMA mode and optionally causes a comparison operation. DMA loads strings at up to two characters per microsecond, and is an important factor in obtaining good system throughput. Without DMA, each string would have to be moved character by character under microprocessor control which would take much longer than the actual Proximity Computation. However, strings can be written directly without using DMA, if desired.

A further advantage of DMA is DMA editing: certain characters are recognized as state indicators, and special state transition characters are optionally generated. DMA also deletes *FILL* (01_{16}) characters. DMA Editing is fully explained in the Advanced Topics chapter. Many applications can use DMA editing to reduce the database size and hence increase throughput.

1.2.2 The Proximity Computer

The second subsystem, the *String Proximity Computer*, hereafter referred to as the Proximity Computer, is the heart of the PF474. Here, a string comparison algorithm computes the Proximity Function, indicating the degree of similarity of two strings stored in the String Memories. Character Parameters control the precise manner in which the degree of similarity is computed. The resulting Proximity Value is a 32–bit binary fraction ranging from zero to one. A Proximity Value of 1 indicates that two words are fully equivalent while a 0 Proximity Value indicates that two strings are completely different (no characters in common, compensation for all characters equal to zero). Values between zero and one denote a uniform range of similarity. The Proximity Function is mathematically elegant, yet its measure of similarity agrees very well with human intuition: the function *sees* the same similarities that people do. These properties make the PF474 a very exciting programming tool.

The utility of the PF474 is enhanced by user–variable parameters and the capability to handle long strings of up to 127 8–bit characters. The Proximity Function allows the user to set *weight, compensation* and *bias* parameters individually for each of the 255 possible characters. These parameters control the importance of each character and allow other customizations of the Proximity Function as described later in this manual. The ability to set parameters allows the PF474 to calculate different values for different applications, making the Proximity Function useful for a wide variety of tasks.

1.2.3 The Ranker Subsystem

The third computational subsystem is the Ranker. The Ranker takes the results of the Proximity Computer and saves the best matches in the Ranked List memory. After doing many successive comparisons, the Ranked List contains information about the comparisons yielding the highest degrees of similarity.

In some cases it may be necessary to retrieve more than the best 16 matches. The Ranker can be used to get the next best 15 matches (and then the *next* 15, and so on) by setting the Next Best flag, initializing the Ranked List, and re–scanning the entire database.

The Ranker saves the Proximity Value and a four byte Internal Record Number (IRN) for each of the up to 16 best (or next best 15) comparisons. The IRN is initialized by the user and automatically incremented by the Ranker subsystem after each comparison, thus generating a unique IRN for each record.

The Proximity Computer and the Ranker operate in a *pipelined* fashion. This means that the Proximity Computer can begin a new Proximity Computation while the Ranker is still ranking the previous Proximity Value. The Ranker works concurrently with the computation of the Proximity Function and thus does not slow down the computational throughput significantly. For most comparisons, the overall throughput of the Ranker is higher than that of the Proximity Computer so that the Proximity Computer will rarely wait for the Ranker. The pipelined structure of the Proximity Computer and Ranker is an important part of the speed advantage gained by using the PF474. Very little extra time is spent in performing ranking. If the ranking were performed by software, the ranking time would be much greater than the computation time!

1.3 APPLICATIONS

Most data from human input, or physical processes comes in an inexact form, which then must be interpreted. The PF474 was designed to assist in solving these non–numeric tasks. In contrast to older methods, the Proximity Function offers a measure of similarity between strings that is both fast and meaningful. Following are some of the more interesting or unusual application of the PF474.

- When the search space is a small to medium sized list (up to about 1000 items), the PF474 is fast enough to provide the appearance of instantaneous response. A PF474 could be used to increase the friendliness of an operating system by instantly finding the intended command (or filename, etc.) from an incomplete or misspelled one.
- The medical and legal professions rely heavily on decisions based on a accumulations of previous experiences, and a quick interactive method for searching case histories would be very useful.
- A DBMS with a PF474 would allow a user to work *with* the computer to find a record instead of operating in an inquire, wait, and try again mode. The user would fill in any of the fields that he knew, and the PF474 would use those fields to find the closest matches. All the fields would not have to be filled in: the desired record will probably show up in the top 16 before most of the fields are filled in.
- Large databases could be searched by having the PF474 (or multiple PF474s) directly read data as it comes off a disk. PF474s could be placed on memory boards with a controller, to provide a search capability with no processor overhead.
- For very large databases, partitioning/clustering techniques could be used to create a fuzzy ISAM file structure. The index blocks would be of very high valence, and the PF474 would search these blocks.
- A natural application for the PF474 is in the area of speech recognition. The PF474's string length of 127 characters allows for a rather detailed representation of an utterance's phonetic structure. The PF474 would then compare the utterance against a list of templates. With an appropriate method for representing phonetic structures, a medium vocabulary speaker–independent recognition system could easily be built.
- Handwriting analysis and recognition is another application suited for the PF474. The various features of a sample of handwriting would be built into a string which would be then compared against a library of templates.

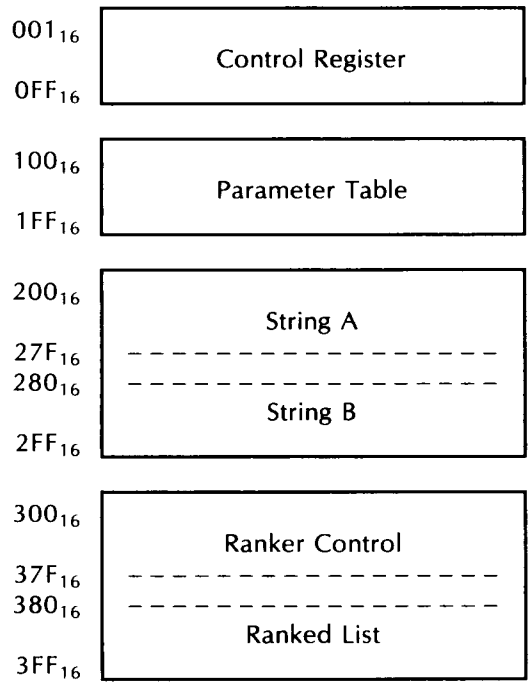
This list is certainly incomplete, especially since the PF474 implements a completely new function, and is not simply a hardware version of existing software. There will be applications that were previously impossible, not simply slow or impractical.

The following list has some general hints and suggestions for PF474 applications.

- Typically, the ordering of the best matches will be of importance, and the actual Proximity Values will be of little significance, though a large gap in Proximity Value between entries usually indicates a large difference in similarity.
- Greater throughput can be realized by putting multiple PF474's into a system. PF474's can be paralleled to use the same query string on different parts of a database, or multiple PF474's could work on the same database but with different query strings (or different parameter settings).
- Most applications will have static parameter settings, but since the parameters are in RAM, they may be dynamically modified. This corresponds to adaptive pattern recognition and will be a part of very advanced applications.

PROGRAMMING

The PF474 appears to the programmer as a 1024 byte memory region. Not all of these locations are read/write; a few are write only or read only. Many addressable locations have not been assigned yet but are reserved for future expansion.



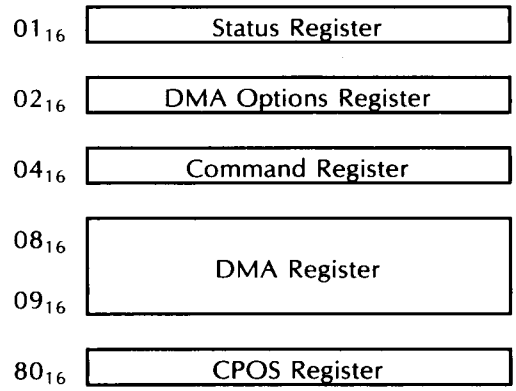
Major blocks of this address space reflect the different functions the PF474 performs. The Control region directs the operation of the chip and provides status information to the host CPU. Parameters for each character in the string alphabet, consisting of **weight**, **bias**, and **compensation**, are stored in the Parameter region (a 256 byte RAM). These are referenced by the Proximity Computer during its calculations. The Proximity Function is calculated for the two strings stored in the String region (another 256 byte RAM). The result is compared with a list of previous results, and stored in a ranked list contained in the Ranker region. The Ranker region also contains additional status and control registers useful in manipulating this ranked list.

This chapter first describes in detail each of these sections. Then, there are program outlines for some typical applications. The next chapter describes in detail DMA editing capabilities, precise restrictions on parameter settings for long strings, software ranking, and throughput calculations.

2.1 THE CONTROL SECTION

The Control section consists of five registers. It occupies addresses $0–FF_{16}$, although only a few addresses are actually assigned at this time. Any unused location should be considered reserved for possible future use.

The Status, DMA Options, Command and DMA Registers are accessible during all phases of a Proximity Computation and Ranking except during DMA. The Clear Position Index Register (CPI) is write–only, and should be accessed only when the SBSY status bit reads zero, as explained below.



2.1.1 The Status Register

The Status Register, a read–only register, contains bits which indicate that the Ranker and/or Proximity Computer are busy. In order to avoid contention after a computation has been initiated, the Status Register should be checked before accessing the the PF474. To determine chip status, a program should read from location 01_{16}.

The Status Register

	BUSY	0	0	0	0	0	0	SBSY
01_{16}	7	6	5	4	3	2	1	0

The low order bit SBSY is set when a Proximity Computation is in progress. The Proximity Computation uses the String Memories and the Symbol Parameter Table; it is not permissible to access these memories when SBSY is set.

The high order bit BUSY is set when either a Proximity Computation or a Ranking is in progress, indicating that the Ranker section is in use. When either the Ranker or the Proximity Computer is busy, it is not permissible to access the Ranked List or the Ranker Control registers.

The unused bits of the Status Register are returned as zero.

The following table summarizes the 3 possible values of this register.

Register Contents (Binary)	PF474 Status	Accessibility		
		String	Parameter	Ranker
00000000	Idle	Yes	Yes	Yes
10000001	Proximity Computer busy	No	No	No
10000000	Ranker busy	Yes	Yes	No

The significance of the (binary) 10000000 value is that a program may load a new string and initiate a new Proximity Computation even though the PF474 is still ranking the last one. Thus, performance may be greatly enhanced by utilizing the pipeline design of the Proximity Computer and Ranker.

2.1.2 The DMA Options Register

The purpose of the DMA Options Register is to specify several parameters for the DMA process. It is a read/write register with the following layout:

The DMA Options Register

	DMGO	SEL	ENGB	ENGA	DELB	DELA	DDPB	DDPA
02_{16}	7	6	5	4	3	2	1	0

When set, DMGO (DMA & GO), bit 7 of the DMA Options Register, instructs the PF474 to automatically clear the position index before each DMA transfer and invoke a Proximity Computation at the completion of that DMA transfer. This feature allows a very tight software loop to control the PF474. When a DMA operation is initiated (See the DMA Registers) while DMGO is set, the following sequence of events occurs:

- SBSY of the Status Register is checked and, if set, the PF474 waits until it is reset
- The String Position Register is cleared (same as writing to the CPI Register)
- A DMA operation is performed
- A Proximity Computation/Ranking is initiated (Same as the GO signal to the Command Register)

By using DMGO, a program can ignore the status of the PF474 and just initiate operations as quickly as it can (assuming DMA operation stops the processor). It is only after the final operation that the program needs to check the Status Register.

The SEL bit of the DMA Options Register SELects String A or String B to receive the DMA data. A 0 selects String A, 1 selects String B.

The remaining bits of the DMA Options Register control the editing of special symbols during the DMA transfer as described in the Advanced Topics section.

2.1.3 The Command Register

The Command Register

	-	-	-	-	-	-	RST	GO
04_{16}	7	6	5	4	3	2	1	0

The Command Register is used to start a Proximity Computation/Ranking or to cause a software reset. It is a write–only register. The GO bit is used to start the PF474 in cases where DMGO is not used (see the DMA Options Register). The RESET bit is used to perform a software reset (same function as hardware reset pin). Specifying both GO and RESET is a programming error. RESET aborts any computation in progress and returns the PF474 to an idle state within three clock cycles. Note that this will scramble the Ranked List until the next Proximity Computation is performed. In many applications the Command Register may be totally unused since the PF474 may be reset at power–on time and the DMGO bit may be set in the DMA Options Register.

2.1.4 The DMA Register

The DMA Register

08_{16}	A7	A6	A5	A4	A3	A2	A1	A0
09_{16}	A15	A14	A13	A12	A11	A10	A9	A8
	7	6	5	4	3	2	1	0

The DMA Register occupies locations 8 and 9 and holds a 16 bit address. The PF474 internally maintains a write access flag for each of these two bytes. When both flags are set, indicating that the program has completed writing both bytes of the address, DMA is automatically invoked. After a DMA operation, this register contains the address immediately following the NULL which terminated DMA, and the internal flags are reset.

During a DMA operation, a transfer of data takes place, with the length determined entirely by the position of the NULL character. It is an error to perform DMA such that the string, minus FILL characters, plus generated transition characters (see DMA Editing), minus deleted state characters (see DMA Editing), plus terminating NULL, is longer than 128 characters.

2.1.5 The CPI Register

The CPI Register

80_{16}	-	-	-	-	-	-	-	CLR
	7	6	5	4	3	2	1	0

The PF474 contains an internal register called POS which is used during DMA as a position index. Writing a one (actually anything) to CPI (Clear Position Index) zeros the POS Register. The CPI Register need not be accessed if the DMGO option of the DMA Options Register is used. To illustrate its use, suppose that the string *pic* is stored in external memory at address 100_{16} and the string *tur* is stored at 200_{16}, and the string *es* is stored at 300_{16}. Then the following register operations would be needed to load the string *pictures* into PF474 String Memory:

- Wait until SBSY in the Status Register is zero
- Write 1 to CPI
- Write 100_{16} to the DMA Register
- Write 200_{16} to the DMA Register
- Write 300_{16} to the DMA Register

Note that the CPI Register must not be written to if SBSY is set. CPI is a write–only register.

2.2 THE PARAMETER SECTION

The Parameter section is a 256 byte memory region which contains one byte for each possible character describing the attributes of that character. These attributes are: weight, bias and compensation.

The **weight** of a character is a direct measure of the importance or significance of a character. A character with weight 6 is twice as important as a character of weight 3 (which is three times as important as a character of weight 1) in determining the similarity of two strings. The weight of a character varies between 0 and 7 inclusive. A character of weight 0 is almost ignored, but does affect similarity by taking up a position in the string.

The **bias** of a character can be set from −2 to +1. A negative bias means that a character is more important near the beginning of a word than near the end of the word. A positive bias means the opposite. Thus, if all characters are uniformly biased negatively, the Proximity Computation will attach more importance to words being similar near the beginning and less importance to similarity near the end of the words.

The **compensation** of a character can be set from a low of 0 to a high of 7. Compensation distinguishes two different kinds of dissimilarity between characters: two words may differ by having a given character in one word but not the other, or by having a character in a different position. The compensation value of *unmatched* characters (but not characters that are just in the wrong position) is added in during the Proximity Computation. High compensation (relative to weight) makes the Proximity Function more tolerant of dropped or missing characters.

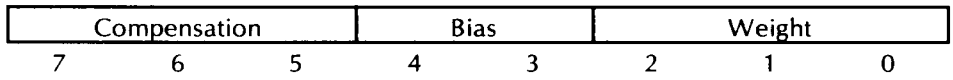
The Parameter section occupies addresses 100_{16}–$1FF_{16}$ and consists of 256 *Symbol Parameters*. Each Symbol Parameter controls the manner in which the PF474 processes a particular symbol. For example, the thirtieth Symbol Parameter has address $100_{16}+1E_{16}=11E_{16}$ and affects processing of the symbol having decimal value 30 ($1E_{16}$). Thus it is possible to individually specify processing parameters for each symbol used in a given application. The NULL symbol also has a Parameter byte assigned to it, but the NULL is not a valid character. It is only a string terminator.

The contents of Parameter Memory are not predictable at power up time and therefore should be initialized before the PF474 is used. However, there is no need to load Symbol Parameters corresponding to symbols that are not used in a given application. Asserting PF474 Reset has no effect on Parameter Memory.

There are restrictions on setting the Symbol Parameters that are explained later. Violation of these restrictions may result in an undetected erroneous value from the Proximity Computation. Therefore, it is the programmer's responsibility to ensure that Symbol Parameters are in the correct range.

Each Symbol Parameter is divided into three separate bit–fields as shown below:

Symbol Parameter Layout



2.2.1 Weight

Weight is a three bit unsigned binary field used to adjust the weight (intuitiveness, importance) of a symbol. Weight can have 8 possible values encoded with bit 0 being the least significant bit.

Seven (binary 1 1 1) is the maximum weight while a weight of zero (0 0 0) causes the PF474 to virtually ignore the symbol. However, even weight zero symbols occupy space in a string and therefore affect to some extent the Proximity Computation.

2.2.2 Bias

Bias is a two bit signed two's complement binary field that may be used to adjust the positional sensitivity of the Proximity Computation both for matching and for non–corresponding symbols. Negative bias values increase sensitivity towards the beginning of the string, reducing linearly with position. Positive bias values increase sensitivity towards the end of the string, increasing linearly with position. A bias of zero is the neutral setting.

There is a simple restriction on the setting of bias: the sum of bias and weight must fall in the range 0–7. For example, if bias is set to –1, the range of legal weights is 1–7. If the bias is 1, the range of legal weights is 0–6.

Bit–4	Bit–3	Bias
1	0	–2
1	1	–1
0	0	0
0	1	1

2.2.3 Compensation

Compensation is a three bit unsigned binary field, with bit 5 being the least significant bit, that is used to adjust overall sensitivity of the PF474 to non–corresponding symbols. When set to zero the PF474 is least tolerant of non–corresponding symbols. In this case the result of comparing *a* with *b* would be zero. Raising compensation relative to weight increases tolerance of missing characters. The extreme case occurs when compensation is equal to weight. For example, if this extreme case occurs for symbols 'a' as well as 'b' then the result of comparing *a* with *b* would be unity! It is useful to view compensation in the following general way: with compensation set to zero, the Proximity Function is more sensitive to missing characters than to errors in positioning; raising compensation/weight ratios gives more importance to position by decreasing the sensitivity to missing characters. This is best illustrated by an example. Consider the two cases of:

aligned compared to *alined*
and
aligned compared to *alinedg*

If **g** has a low compensation/weight ratio, the second comparison will yield a higher degree of similarity than the first comparison. This is because **g** is entirely missing from the second word in the first comparison, but is in the wrong position in the second comparison. If the character **g** has a high compensation value relative to its weight, the first comparison will yield a higher degree of similarity. Thus, high compensation corresponds to a high tolerance for missing characters. In this example, raising the compensation made the Proximity Function correspond more closely to our own notion of similarity: a single omission is of less significance than a character misplaced three positions.

Compensation is restricted in that it should never be greater than weight, nor greater than the sum of weight and bias.

Setting all Symbol Parameters to 1_{16} (compensation = 0, bias = 0, weight = 1) is perhaps the most basic setting. Adjusting the Symbol Parameters is a powerful method for adapting the Proximity Computation to specific applications. In most cases this adjustment will occur during a product's design cycle and then be frozen (or static) for a given application during product operation. Very sophisticated applications may require dynamic adjustment of the Symbol Parameters in real–time.

2.2.4 Restrictions

This is a brief summary of the PF474 Symbol Parameter value restrictions:

- Bias + Weight should always be in the range 0–7
- Compensation must be no greater than Weight or Weight + Bias
- Total weight must be less than 32,768 (affects only strings longer than 67 characters).

The *total weight* limit is a somewhat involved restriction on the choice of weight and bias which is necessary to avoid internal arithmetic overflow in the Proximity Computer. The following quick reference table gives the maximum string length for various settings of bias and weight:

		W_{max}							
		0	1	2	3	4	5	6	7
B_{max}	1	**127**	**127**	113	96	84	76	70	
	0	**127**	**127**	**127**	104	90	80	73	67
	−1		**127**	**127**	113	96	84	76	70
	−2			**127**	**127**	104	90	80	73

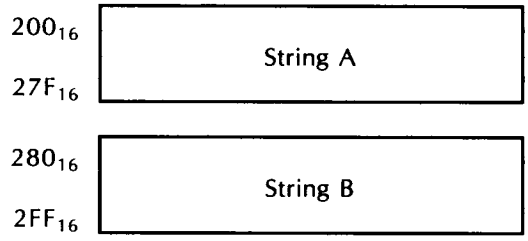
Given the maximum weight and bias to be used, look up their intersection in the table to find the maximum legal string length. For example, if the maximum bias value used in a given application is zero and the maximum weight is 6, under all circumstances, strings of length 73 or less are legal. The blanks correspond to illegal combinations of bias and weight (bias plus weight must be in the range: 0 – 7).

Calculation of total weight is explained in the Advanced Topics section.

2.3 THE STRINGS SECTION

The Strings section holds the two strings to be compared. Each string can be up to 127 bytes long plus a NULL terminating character. The strings may be loaded by directly writing into the String Memories, or they may be loaded via a DMA operation. After two strings are loaded, the Command Register may be used to start the comparison operation.

Generally, one of the strings in the String Memories is loaded as a *query* string. Then a large number of *database strings* are written, one by one, into the other string memory and compared against the first string. It does not matter which string memory is used for which string: the Proximity Function is symmetric with respect to its arguments.



The first byte of each string is written to the lowest memory address in its region.

WARNING: If there is no NULL terminator, the PF474 will not behave properly. The NULL is not a valid character; it is only to be used as a terminator.

The PF474 recognizes *only* the most recent NULL written as the end of string terminator. There must not be any NULLs within the string. Thus, the following sequence of program writes is improper:

- Write 'c' to location 200_{16}
- Write 'a' to location 201_{16}
- Write 't' to location 202_{16}
- Write NULL to location 203_{16}
- Write NULL to location 204_{16}

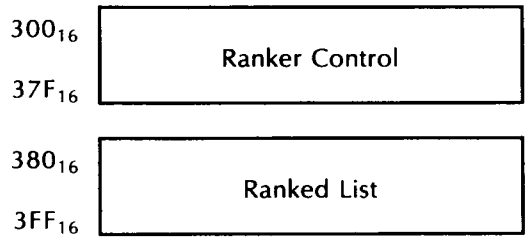
But the following sequence does not constitute a programming error:

- Write 'c' to location 200_{16}
- Write 'a' to location 201_{16}
- Write 't' to location 202_{16}
- Write NULL to location 204_{16}
- Write NULL to location 203_{16}

This sequence is acceptable because the PF474 remembers 203_{16} as the last location to which a NULL was written, which is the proper location for the string terminator. This feature may also be used to slightly alter or shorten strings already loaded in String Memory by simply writing a new NULL at the desired location.

2.4 THE RANKER SECTION

The primary purpose of the Ranker is to maintain up to 16 slots corresponding to the *best matches* that the PF474 has encountered. It occupies addresses 300_{16}–$3FF_{16}$ and consists of two major subsections of 128 bytes each as shown below:



The Ranked List contains the slots referred to above and the Ranker Control consists of control and status registers.

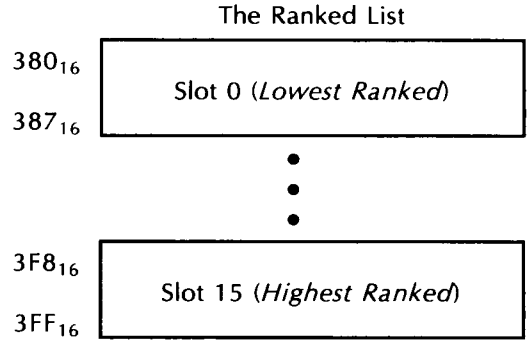
The Ranked List memory is a table with 16 entries (slots), containing the best results of the string comparisons. Each entry holds the result of a Proximity Computation and the Internal Record Number (IRN) of the string matched.

The result of the Proximity Computation is a 32–bit binary fraction called the Proximity Value. The Proximity Value ranges from 0 to 1 ($0000\ 0000_{16}$ to $FFFF\ FFFF_{16}$), with a high value denoting close similarity and a low value denoting dissimilarity. The Proximity Value should be thought of as a pure mantissa number; for example, $C000\ 0000_{16}$ is $0.C0000000_{16}$ in base 16, equal to 0.75 in base 10. The least significant bit should be viewed as extending to the right to infinity, making a Proximity Value of 1 ($FFFF\ FFFF_{16}$), when written as a pure mantissa number, equal to $0.FFFFFFFFFF..._{16}$ which equals 1.

The 4 byte IRN uniquely identifies the string which ranked. You will note that the strings which were compared to produce the Proximity Value are NOT stored in the ranker slot. Instead an IRN, typically a database record number, is stored in the ranker slot. The Next IRN Register is automatically incremented by 1 for each record compared, whether or not the word is ranked. The Next IRN Register can be re–initialized directly during a series of comparisons, so that the software may conveniently retrieve the words which ranked. The BUSY bit of the Status Register must be clear (indicating an idle Ranker) when the Ranked List or any Ranker Control register is accessed.

2.4.1 The Ranked List

The Ranked List occupies locations 380_{16} through $3FF_{16}$ (128 bytes) and consists of 16 slots of 8 bytes. The lowest ranked slot is the first slot, occupying addresses 380_{16} through 387_{16}. The highest ranked slot is the one corresponding to the value of the Size Register (see below). Thus, if the Size Register contained $0F_{16}$ (15), the highest ranked slot would be slot 15 occupying addresses $3F8_{16}$ through $3FF_{16}$.



Each slot holds a 32–bit Proximity Value as well as a 32–bit IRN. Each of these occupies 4 bytes with the low order byte in low memory. The diagram below details this layout calling the 32–bit Proximity Value *P* with bits 0–31 (0 is low order) and calling the 32–bit IRN *I* with bits 0–31 (0 is low order):

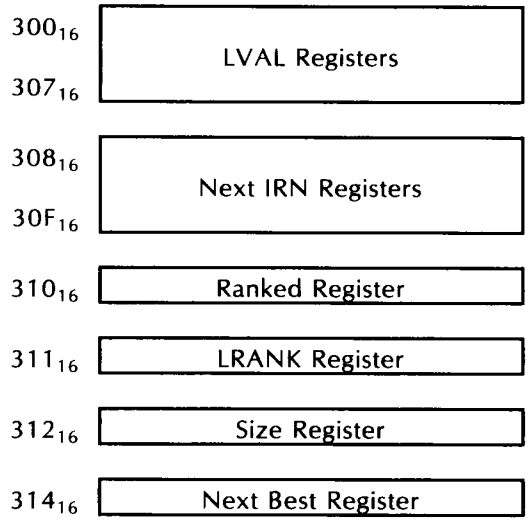
Ranker Slot Layout

Byte 0	P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0
Byte 1	P_{15}	P_{14}	P_{13}	P_{12}	P_{11}	P_{10}	P_9	P_8
Byte 2	P_{23}	P_{22}	P_{21}	P_{20}	P_{19}	P_{18}	P_{17}	P_{16}
Byte 3	P_{31}	P_{30}	P_{29}	P_{28}	P_{27}	P_{26}	P_{25}	P_{24}
Byte 4	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0
Byte 5	I_{15}	I_{14}	I_{13}	I_{12}	I_{11}	I_{10}	I_9	I_8
Byte 6	I_{23}	I_{22}	I_{21}	I_{20}	I_{19}	I_{18}	I_{17}	I_{16}
Byte 7	I_{31}	I_{30}	I_{29}	I_{28}	I_{27}	I_{26}	I_{25}	I_{24}
	7	6	5	4	3	2	1	0

The ranker memory MUST be initialized after power–up and before the first comparison is initiated. This normally consists of setting the entire list to zero. In addition it should be initialized after a software initiated reset except in cases where the Status Register was examined and revealed that the ranker was not busy. In this instance, the reset will temporarily scramble the Ranked List until the next Proximity Computation is performed. This will restore the Ranked List to its properly ordered form.

2.4.2 Ranker Control

This section describes the **Ranker Control** portion. The map below details its layout:



Note that some locations are not currently used in the PF474's memory space. All unused locations are reserved for possible future use.

When a record is ranked, the Proximity Value is saved in the LVAL Registers. There are eight LVAL Registers occupying locations 300_{16} through 307_{16}. Each is a read/write register, although there is is generally no reason to ever write to them. LVAL stands for *Last Value*. The purpose of these registers is to permit a program to determine the value resulting from the last Proximity Computation performed (only if the last word examined ranked). This register is not used in most applications since the Proximity Values of most interest are those which are stored in the ranked list. LVAL will **NOT** contain a valid Proximity Value if the current comparison didn't rank.

Building a 32–bit Proximity Value from the contents of these registers is less than straightforward since only one nibble (4 bits) is obtained from reading a single register. Calling the 32–bit Proximity Value *P* with bits 0–31 (0 is low order) we have the following layout:

LVAl Layout

	7	6	5	4	3	2	1	0
300_{16}	0	0	0	0	P31	P30	P29	P28
301_{16}	0	0	0	0	P27	P26	P25	P24
302_{16}	0	0	0	0	P23	P22	P21	P20
303_{16}	0	0	0	0	P19	P18	P17	P16
304_{16}	0	0	0	0	P15	P14	P13	P12
305_{16}	0	0	0	0	P11	P10	P9	P8
306_{16}	0	0	0	0	P7	P6	P5	P4
307_{16}	0	0	0	0	P3	P2	P1	P0

The Next IRN Registers together contain the next 32–bit IRN to be used by the PF474. There are eight Next IRN Registers occupying locations 308_{16} through $30F_{16}$. Each is a read/write register. **Next IRN** stands for *Next Internal Record Number*. Just before a search, or perhaps a phase of a search begins, the program writes a 32–bit value into these registers, one nibble at a time as indicated by the table below. After each Proximity Computation is performed, the Ranker circuits of the PF474 attempt to place it in the ranked list. If the record is ranked, then the value of the Next IRN Register is copied into the proper ranker slot along with the Proximity Value. Whether or not the computation ranks, but after the attempt has been made, the Next IRN value is incremented as a 32–bit unsigned binary quantity. Thus Next IRN is a counter which uniquely identifies each comparison. Calling the 32–bit IRN *N* with bits 0–31 (0 is low order) we have the following layout:

Next IRN Layout

	7	6	5	4	3	2	1	0
308_{16}	0	0	0	0	N31	N30	N29	N28
309_{16}	0	0	0	0	N27	N26	N25	N24
$30A_{16}$	0	0	0	0	N23	N22	N21	N20
$30B_{16}$	0	0	0	0	N19	N18	N17	N16
$30C_{16}$	0	0	0	0	N15	N14	N13	N12
$30D_{16}$	0	0	0	0	N11	N10	N9	N8
$30E_{16}$	0	0	0	0	N7	N6	N5	N4
$30F_{16}$	0	0	0	0	N3	N2	N1	N0

The Size Register is used to set the size of the PF474's ranked list. It is a read/write register located at location 312_{16}. Only the lower four bits are used:

Size Register

	7	6	5	4	3	2	1	0
312_{16}	0	0	0	0	S3	S2	S1	S0

The size of the ranked list can be set to any value from 1 to 16, by writing one less than the desired size into the Size Register. For example, to set the size to 11 (decimal) one writes $0A_{16}$ (10 decimal) which is 00001010 (binary). The upper four bits are ignored during a write but are returned as zero during a read operation. Normally the Size Register is just set to its maximum value of $0F_{16}$. Lower values may slightly improve the PF474's overall throughput but in general do not do so significantly. However, if the user code examines the RANKED and LRANK Registers after each computation to save the actual strings, a smaller ranked list might significantly improve performance by reducing the frequency with which computations rank and cause the software to save the string.

When read, the LRANK Register provides the position number in the ranked list that received the last ranked record. If a record does not rank the LRANK Register is unchanged (see the RANKED Register below). It is normally used as a read–only register but can be written to. The upper four bits are returned as zero.

LRANK Register

311_{16}	0	0	0	0	P3	P2	P1	P0
	7	6	5	4	3	2	1	0

When read, the RANKED Register returns high values (00001111 binary) if the last Proximity Computation ranked and returns zero otherwise.

RANKED Register

If last record ranked:

310_{16}	0	0	0	0	1	1	1	1
	7	6	5	4	3	2	1	0

If last record did not rank:

310_{16}	0	0	0	0	0	0	0	0
	7	6	5	4	3	2	1	0

NB is a write–only register located at address 314_{16} which establishes the ranker's general mode as either normal or *next best*. In normal mode the ranker selects the best matches it encounters and ranks them. Next–best mode is used to effectively extend the size of the ranked list by locating up to 15 next best matches. Typically the first time the PF474 processes a list of records it is used to locate the best 16. If a longer list is desired, the application program copies the IRN and the Proximity Value ranked lowest during the first pass into the highest ranked slot, clears all the other slots to 0, writes a one into the NB Register, resets the Next IRN Register, and then uses the PF474 to reprocesses the record list. The list will contain the next best 15 matches. This process can be repeated to get the next 15 and so on. In our example we assumed the Size Register contained $0F_{16}$ (selecting a ranked list size of 16) but next best processing will work for any ranked list length above 1. It should also be noted that the Size Register can be set differently during the first pass and subsequent passes. For example, the Size Register could be set to 05_{16} for the first pass and to $0F_{16}$ for the second. The NB Register is not affected by a RESET and, therefore, must be initialized before the first Proximity Computation is initiated.

When writing to the NB Register only the lowest bit is important. '1' selects next–best mode and '0' selects normal mode.

Next Best Register

314_{16}	-	-	-	-	-	-	-	NB
	7	6	5	4	3	2	1	0

2.5 A SAMPLE PROGRAM

The program outline that follows is a simple database search, without using DMA to load strings. This could be used to scan a lexicon to check the spelling and find the closest matches to the given word. If the word is correctly spelled (it is found in the lexicon), the best match will be the word itself, and it will be at the top of the list with a Proximity Value of 1.

Application Initialization:

This initialization is required once for a given application.

- Symbol Parameters set to 01_{16}
- Size of Ranked List set to 16 ($0F_{16}$)
- Next Best mode negated (set to 0)

Search Initialization:

This initialization is required at the beginning of each list search.

- Next IRN initialized by writing all 0's
- Ranked List initialized by writing all 0's
- Query string written to String A

Search:

This is the procedure to be repeated for each item in the database.

- Wait for Status bit 0 (SBSY) to read 0
- Next lexicon entry written to String B
- Start the Proximity Computation and Ranking by writing a GO command to the Command Register

Termination:

When all database entries have been compared to the query string, read the results from the Ranker.

- Wait for Status Register to indicate that the Ranker is done (BUSY = 0)
- Read the Ranked List

The program then uses Internal Record Numbers (IRNs) to retrieve ranking strings from the database, or stores the IRNs for later reference.

2.5.1 Programming the PF474 in DMA Mode

In some applications, CPU control of data transfer is adequate (or the processor is generating the database as it goes along – decompressing the database, for example). When greater speed is desired, and the database is directly available in memory, the PF474 may be operated in Direct Memory Access (DMA) mode. In this mode, the PF474 takes control of the system bus to transfer data as rapidly as possible.

In addition to simply transferring data, the PF474 is capable of building strings from segments stored in separate locations, and editing in or out certain special symbols, augmenting the pattern matching power of the PF474. The following example describes the simplest use of the DMA feature, data transfer. During the DMA transfer, the processor is assumed to be halted.

Application Initialization:

This initialization is required once for a given application.

- Symbol Parameters set to 01_{16}
- Size of ranked list set to 16 ($0F_{16}$)
- Next best mode negated (set to 0)

Search Initialization:

This initialization is required at the beginning of each list search.

- Next IRN initialized by writing all 0's
- Ranked List initialized by writing all 0's
- Query string written to String A
- DMA Options Register set to $C0_{16}$. This instructs the PF474 to direct DMA transferred strings to String B, start Proximity Computations as soon as the transfer is complete, and to do no special DMA editing.

Search:

This is the procedure to be repeated for each item in the database.

- Write address of next lexicon word (obtained by reading the DMA Register) to DMA Register as soon as the previous DMA transfer is complete

Termination:

When all database entries have been compared to the query string, read the results from the Ranker.

- Wait for Status Register to indicate that the Ranker is done (BUSY = 0)
- Read the Ranked List

If the processor is not halted during DMA transfers, the program will have to determine that one DMA transfer has terminated before starting the next one.

ADVANCED TOPICS

3.1 LONG STRING PARAMETER RESTRICTIONS

This section describes the restriction on the Total Weight of strings that applies when the strings are longer than 67 characters. To avoid an internal arithmetic overflow condition within the PF474, a rule must be obeyed when setting the weight and bias fields. As previously noted, this rule only restricts the Symbol Parameters when string length exceeds 67 characters (not including the NULL terminator). In other words, if an application will never involve strings of length greater than 67, then the choice of weight and bias must only meet the first two restrictions as explained in the Parameter section – Bias + Weight must be in the range 0–7, and Compensation must be no greater than Weight or Weight + Bias.

Presented here is a rather involved but precise mathematical formulation of the restriction. Then we will reduce this to a simple design table. Even if strings longer than 67 characters are used, the table is all that is necessary to check in the vast majority of cases. The mathematical formula is presented for completeness, and to make the table easier to understand.

We begin by defining the *total weight* (TW) of a string:

$$TW = \sum_{i=1}^{L} iB_i + W_i\,(L+1)$$

In the equation above B_i refers to the bias of the ith symbol in the string, W_i refers to the weight of the ith symbol and L refers to the length of the string.

To avoid overflow, we must have:

$$TW < 32{,}768$$

Once weights and biases are set, we could use this equation to predict whether a given string will cause an overflow. However, in most applications, one doesn't know the strings in advance. By making a few assumptions about the nature of the strings used, we can reduce this to a usable table.

Let us denote by W_{max} the maximum weight we intend to use in a given application. We will assume the worst case, that it is possible for the entire string to consist of symbols of this weight. Therefore, we have:

$$TW = W_{max}L(L+1) + \sum_{i=1}^{L} iB_i$$

In the same way, let us denote by B_{max} the maximum bias we intend to use in an application. Again, assume that it would be possible for the entire string to consist of symbols with this bias. Now we have:

$$TW = L(L+1)(W_{max} + B_{max}/2)$$

We now use this relation to generate the quick reference table that also appears in the Parameter section:

		W_{max}							
		0	1	2	3	4	5	6	7
B_{max}	1	**127**	**127**	113	96	84	76	70	
	0	**127**	**127**	**127**	104	90	80	73	67
	−1		**127**	**127**	113	96	84	76	70
	−2			**127**	**127**	104	90	80	73

Given the maximum weight and bias to be used, look up their intersection in the table to find the maximum legal string length. Blanks correspond to illegal combinations of bias and weight (bias plus weight must be in the range: 0 – 7). For example, if the maximum bias value used in a given application is zero and the maximum weight is 6, then under all circumstances, strings of length 73 or less are legal.

3.2 DMA EDITING

To understand this optional feature of the DMA process, we must first examine the alphabet of the PF474. The PF474 alphabet contains 256 one byte symbols. One useful interpretation of this alphabet, which will be assumed in the examples in this section, assigns the first 128 symbols (except NULL) to be equivalent to the ASCII code. The remaining 128 characters (those with the most significant bit set) may be user–assigned to have special properties during DMA transfer. Each of the 255 alphabet symbols may be independently used to fine–tune the Proximity Computation by adjusting that symbol's parameters.

If DMA is not used, only the NULL symbol (00_{16}) has a special meaning as a string terminator. The remaining 255 symbols are valid input string symbols for Proximity Computations and may have their parameters set individually.

If DMA is used, the FILL character (01_{16}) also becomes special: it will always be stripped out by the DMA process. The 128 symbols with the most significant bit set may also be designated to be special symbols during DMA, with particular editing properties specified by the user. These properties depend on the *group* and *type* of each special symbol, and are turned on or off by the settings of the corresponding bits of the DMA Options Register. Automatic special symbol DMA editing is an advanced feature which enhances throughput performance for some applications.

The precise definition of each symbol type appears in the following table. The examples which follow explain their functions. A complete table of special symbols appears in the appendix.

Binary Symbol Value	Name	Description
00000000	NULL	Used to terminate PF474 strings. This symbol is therefore not available for other uses.
00000001	FILL	During DMA this symbol is always deleted, i.e. read from external RAM but ignored. It has no other special properties.
10*xxxyyy*	GA^{xxx}_{yyy}	These are the 64 GA (which stands for *group A*) symbols. They are treated specially only during DMA and then only if the DMA Options Register so indicates. When $xxx=yyy$ we call the resulting symbol SA_{xxx}. For example, when $xxx=yyy=101$ we have SA_5. SA stands for *state symbol/A-group*. When $xxx \neq yyy$ we call the resulting symbol TA^{xxx}_{yyy}. TA stands for *transition symbol/A-group*.
11*xxxyyy*	GB^{xxx}_{yyy}	These are the 64 GB (which stands for *group B*) symbols. They are treated specially only during DMA and then only if the DMA Options Register so indicates. When $xxx=yyy$ we call the resulting symbol SB_{xxx}. For example, when $xxx=yyy=101$ we have SB_5. SB stands for *state symbol/B-group*. When $xxx \neq yyy$ we call the resulting symbol TB^{xxx}_{yyy}. TB stands for *transition symbol/B-group*.

Of the remaining six bits of the DMA Options Register, three apply exclusively to Group A symbols (ENGA, DELA, and DDPA), and three apply exclusively to Group B symbols (ENGB, DELB, and DDPB). All details of the behavior of ENGA, DELA, and DDPA apply also to their *B* counterparts; their functions are entirely analogous. ENGA stands for ENable Group A; this bit enables the generation of TA symbols based on SA symbols. DELA stands for DELete group A state symbols. DDPA stands for Delete DuPlicate group A state symbols.

It is worthwhile to note that the use of ENGA/B may lengthen a string and that the use of FILL characters will shorten the string as it is transferred from external memory to the PF474. It is the user's responsibility to see that this never results in a string of length greater than 127 (excluding the NULL terminator).

The examples below demonstrate the editing which modifies an external string as it is transferred to an internal string. The strings are shown as ASCII and special group A and group B symbols with their hexadecimal values.

If DELA is set, all SA symbols are filtered out (stripped/deleted) during the DMA operation. In the example below there are three SA symbols which are deleted. There is also a FILL symbol which is deleted. For example, if the DMA Options Register contains *xx*000100:

External String	(Hex)	Internal String	(Hex)
a	61	a	61
SA_0	80		
b	62	b	62
SA_3	9B		
c	63	c	63
SA_7	BF		
d	64	d	64
FILL	01		
e	65	e	65
NULL	00	NULL	00

If DDPA is set, all repeated (duplicated) SA symbols are filtered out (stripped/deleted) during the DMA operation. For example, if the DMA Options Register contains *xx*000001:

External String	(Hex)	Internal String	(Hex)
a	61	a	61
SA_1	89	SA_1	89
b	62	b	62
SA_3	9B	SA_3	9B
SA_3	9B		
c	63	c	63
SA_7	BF	SA_7	BF
SA_3	9B	SA_3	9B
NULL	00	NULL	00

If ENGA is set, the PF474 will generate TA symbols during DMA as SA symbols are encountered. Each time a SA symbol differs from the previous one, a TA symbol is generated and inserted after the later SA symbol. Note that all strings are considered to start in state 0; the initial SA symbol for state one generates a transition. For example, suppose the DMA Options Register contains *xx*010000:

External String	(Hex)	Internal String	(Hex)
SA_1	89	SA_1	89
		TA_0^1	88
SA_2	92	SA_2	92
		TA_1^2	91
SA_3	9B	SA_3	9B
		TA_2^3	9A
SA_4	A4	SA_4	A4
		TA_3^4	A3
NULL	00	NULL	00

Processing of states and transitions is performed independently for SA and SB; each group set has no influence on the other. Also, since processing assumes strings start in the zero state, the DDPA and DDPB bits cause deletions of initial zero state symbols in the string. These features are illustrated in the following example. Again suppose the DMA Options Register contains *xx*000001:

External String	(Hex)	Internal String	(Hex)
SB_1	C9	SB_1	C9
g	67	g	67
r	72	r	72
SA_0	80		
e	65	e	65
SB_1	C9	SB_1	C9
a	61	a	61
SA_1	89	SA_1	89
t	74	t	74
SB_1	C9	SB_1	C9
e	65	e	65
SB_5	ED	SB_5	ED
s	73	s	73
t	74	t	74
SA_1	89		
NULL	00	NULL	00

The order in which the DMA editing operations for each group are performed is DDP, ENG, then DEL. Therefore, if DEL and ENG are set, transition characters are generated before state characters are deleted. DDP and ENG work independently, since transition characters are not generated for repeated state characters, even if they are not deleted. With the same input string, suppose the DMA Options Register contains *xx*010101:

External String	(Hex)	Internal String	(Hex)
SB_1	C9	SB_1	C9
g	67	g	67
r	72	r	72
SA_0	80		
e	65	e	65
SB_1	C9	SB_1	C9
a	61	a	61
SA_1	89	TA_0^1	88
t	74	t	74
SB_1	C9	SB_1	C9
e	65	e	65
SB_5	ED	SB_5	ED
s	73	s	73
t	74	t	74
SA_1	89		
NULL	00	NULL	00

Many of the above features are combined in the following example. This will be instructive for the reader to study. Suppose the DMA Options Register contains *xx*110111, so that only DELB is not implemented:

External String	(Hex)	Internal String	(Hex)
SB_1	C9	SB_1	C9
		TB_0^1	C8
g	67	g	67
r	72	r	72
SA_0	80		
e	65	e	65
SB_1	C9		
a	61	a	61
SA_1	89	TA_0^1	88
t	74	t	74
SB_1	C9		
e	65	e	65
SB_5	ED	SB_5	ED
		TB_1^5	E9
s	73	s	73
t	74	t	74
SA_1	89		
NULL	00	NULL	00

Note that if one wanted to have only the NULL terminated string "g r e a t e s t " arrive in String Memory, the desired DMA Options setting would be *xx*001100.

Applications may require more symbols than just the 128 ordinary characters, yet reassigning state characters may be undesirable, because of the usefulness of their properties with the editing options. When this is the case, the user may use transition characters as meaningful input symbols.

TA and TB characters which appear in external memory are not affected by any of the editing processes, as the following example shows. Assume the DMA Options Register contains *xx*111111.

External String	(Hex)	Internal String	(Hex)
a	61	a	61
SA_4	A4	TA_0^4	A0
s	63	s	63
TA_6^7	BE	TA_6^7	BE
y	7A	y	7A
SB_3	DB	TB_0^3	D8
o	6F	o	6F
u	75	u	75
TA_6^7	BE	TA_6^7	BE
SB_2	D2	TB_3^2	D3
a	61	a	61
SA_4	A4		
r	72	r	72
e	65	e	65
TA_7^6	B7	TA_7^6	B7
NULL	00	NULL	00

In summary, transition symbol generation is completely controlled by ENGA and ENGB and is unaffected by the settings of DDPA, DDPB, DELA, and DELB. The deletion of state symbols is controlled by the DDP and DEL options, with DEL naturally overriding DDP. Words are assumed to start in state 0, both by the duplicate symbol deletion and transition symbol generating functions. GA and GB character operations are fully independent of each other. TA and TB characters in the external string are never affected, and thus may be used as ordinary characters.

3.3 THROUGHPUT CONSIDERATIONS

The PF474 is typically used to search and locate the best matches of a query string to entries within a database of strings. This section will discuss the throughput rate (in strings examined per second) for such a task, providing a starting point for estimating how much time any such application will take in a given system.

Four stages of data processing are significant to this discussion of throughput. The first is CPU activity: any operations the central CPU has to perform while the database search is in process. These might include system functions unrelated to the PF474, as well as operations which prepare data strings for DMA, including the write operations which initiate the DMA transfer. The second is the DMA transfer, in which the PF474 takes control of the system bus and moves a string into String Memory. The third is the Proximity Computation, in which two strings are compared and a Proximity Value is produced. The fourth is Ranking, in which the most recent Proximity Value is sorted into the correct location in the PF474 Ranker Memory.

The PF474 has a pipeline architecture. Therefore, some of these separate stages are allowed to overlap in time. In particular, CPU functions can run in parallel with the Proximity Computation and ranking. Ranking may also overlap DMA transfer and the Proximity Computation.

In a typical application, the order of events is as follows.

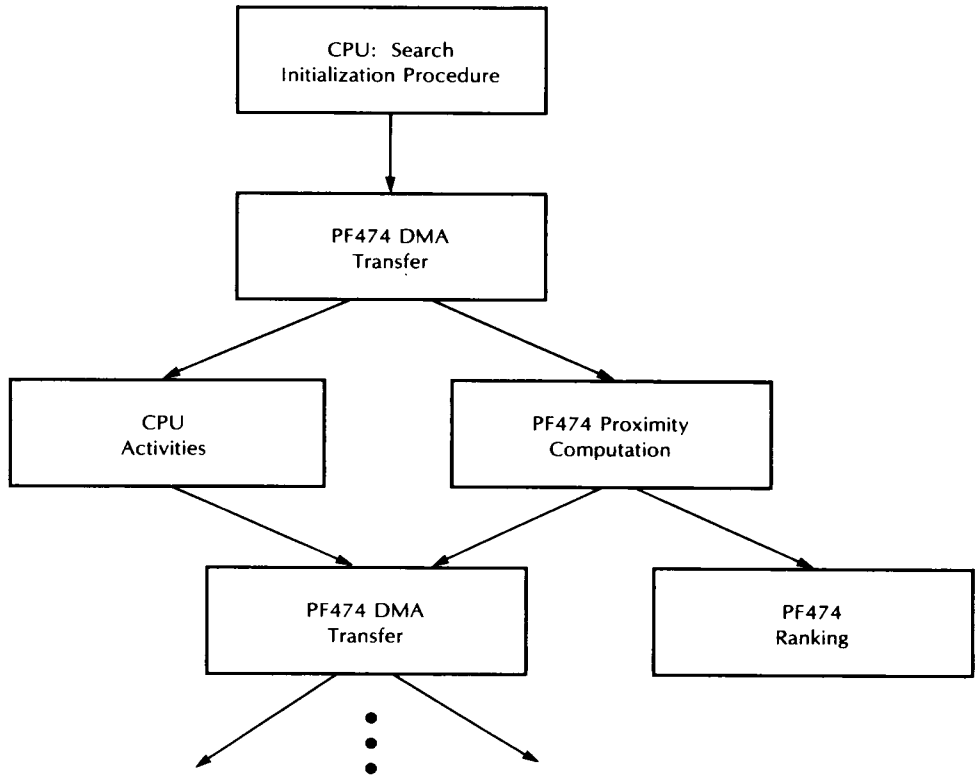


Figure 2. Processing Flow Chart

The CPU first invokes DMA. The end of DMA will initiate the Proximity Computation in parallel with resumed CPU activity. The next DMA transfer will begin immediately after the latest of either the completion of the Proximity Computation, or the CPU's completion of invocation of DMA (writing to location 8 and 9 of the PF474). In addition, the completion of the Proximity Computation will initiate the Ranking process, in parallel with (perhaps) the remainder of the CPU cycle and the DMA transfer. Thus, the repeating cycle for each string is DMA transfer, parallel Proximity Computation and CPU activity, DMA transfer, parallel activity, and so forth, with the Ranker operating entirely in parallel (and thus not affecting throughput) after each Proximity Computation.[1]

1. In rare circumstances, the Ranker may delay the processing of a subsequent string if that string is very short and transferred at the maximum DMA rate, and the previous comparison produced a result which ranked near the top of the list. If this Ranking takes longer than both the new DMA transfer and Proximity Computation, then the termination of that Proximity Computation will be delayed in the pipeline until the Ranker is finished. In the most extreme case, the time for Ranking can be as long as 175 clock cycles; however, the mean time to rank a string in the list is only 12.1 clock cycles.

   This Ranker delay cannot occur for items which do not rank in the top list, and even then, delays will be rare. For databases with thousands or tens of thousands of items, ranking rarely occurs. The actual overhead due to Ranker delay is data and application dependent, and therefore cannot be calculated here. However, in a typical benchmark test with over 4000 strings of mean length of 17, the overhead time for Ranker delay was 0.01 per cent of the total time.

Thus, the time intervals relevant to throughput are the string DMA transfer time (t_{DMA}), CPU preparation time (t_{CPU}), and Proximity Computation time (t_{PXC}). From the diagram above, the time for processing each string t_S is:

$$t_S = t_{DMA} + \max(t_{CPU}, t_{PXC})$$

That is, the processing time is the DMA transfer time plus the time for whichever of the parallel processes takes longest. We now focus on each of these time intervals.

The implementation of the Proximity Function in the PF474 is unique among string matching functions in that the time requirement for the calculation is linear with respect to the length of the strings involved. The average number of clock cycles for each computation is:

$$n_{CYC} = 2L_A + 2L_B + 10.5$$

where L_A and L_B are the lengths of String A and String B (all references to string length in this section do not include the necessary terminating NULL of each string). The 10.5 clock cycles represents the average of a combination of overhead factors, not dependent upon string length. Thus, the time required for each computation is:

$$t_{PXC} = (2L_A + 2L_B + 10.5) \times t_{CYC}$$

where t_{CYC} is the clock cycle time.

DMA input time for each database string is:

$$t_{DMA} = (L + 1) \times t_{DCYC}$$

L is the string length, and t_{DCYC} is the time for each byte transfer. This t_{DCYC} depends on the system implementation; slow memories or timesharing with the CPU may be accommodated with the hardware $\overline{\text{WAIT}}$ line to the PF474, extending the DMA cycles. Without using wait states, t_{DCYC} is just twice t_{CYC}

Processor functions which may run in parallel with the PF474 are only limited in that they must avoid accessing most of the memory space of the PF474. A typical process might prepare for the next DMA transfer by retrieving a block of data from a disk and decompressing or preprocessing this data. The minimum processor function is to write to locations 8 and 9 in the PF474 address space to invoke the next DMA. (This is one of the few PF474 memory locations accessible during Proximity Computations.) This operation may be performed in parallel with the ongoing Proximity Computation; the DMA Controller will wait for the computation to finish before beginning the actual transfer.

As an example of this throughput calculation, we present here the derivation for the maximum throughput rate for a 3 MHz PF474–30. This derivation assumes a 3MHz clock (t_{CYC} is 333 nanoseconds), a system which always invokes DMA transfer as quickly as the PF474 will allow, no DMA timesharing, and memories fast enough to avoid the use of wait states.

This is not merely a theoretical, impossible maximum; systems may easily be constructed which reach it. Interface circuits for existing systems will typically achieve fifty to ninety–five per cent of this maximum throughput.

As the minimum t_{CPU} is 800 nanoseconds (the length of time for writing to locations 8 and 9), and t_{PXC} is always greater than 3.5 microseconds (the time for comparing zero length strings), only the t_{PXC} term applies, and the equation is simply

$$t_S = t_{PXC} + t_{DMA}$$

For DMA transfer without wait states,

$$t_{DMA}=(2L+2)\times t_{CYC}$$

For the case where both strings are length L,

$$t_{PXC}=(4L+10.5)\times t_{CYC}$$

Thus,

$$t_S=(4L+10.5+2L+2)\times t_{CYC}$$

$$=(6L+12.5)\times 333 \textit{ nanoseconds}$$

$$=1988L+4162 \textit{ nanoseconds}$$

This yields the following table:

Maximum PF474 Throughput

Strings per second (thousands)

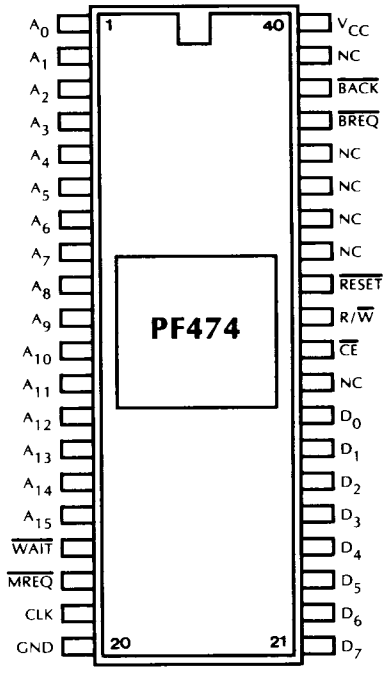
String Length	PF474–25 2.5 MHz	PF474–30 3.0 MHz	PF474–36 3.6 MHz	PF474–40 4.0 MHz
3	81.9	98.3	118.0	131.1
4	68.4	82.1	98.6	109.5
5	58.8	70.5	84.7	94.1
6	51.5	61.8	74.2	82.4
7	45.8	55.0	66.0	73.3
8	41.3	49.5	59.5	66.1
9	37.5	45.1	54.1	60.1
10	34.4	41.3	49.6	55.1
12	29.5	35.5	42.6	47.3
15	24.3	29.2	35.1	39.0
20	18.8	22.6	27.1	30.1
25	15.3	18.4	22.1	24.6
30	12.9	15.5	18.7	20.7
40	9.9	11.8	14.2	15.8
60	6.7	8.0	9.6	10.7
90	4.5	5.4	6.5	7.2
127	3.2	3.8	4.6	5.1

HARDWARE INTERFACING

4.1 PF474 SIGNAL AND INTERFACE DESCRIPTION

The PF474 is electrically interfaced much like a 1024 by 8 static RAM. It has a data and address bus, a chip enable input, and a read/write control input line. In addition, it requires a single phase clock and a reset line. Three pins control the DMA operation, using a simple and widely used protocol.

The PF474 uses its address bus in two ways. When the PF474 is being accessed as memory (non DMA mode), the low order 10 bits, A0–A9, are used as inputs to address a particular location within the PF474. The high order 6 bits, A10–A15, are irrelevant and held in the 3-state *off* condition. In DMA mode, the entire 16 bits, A0–A15, are used as outputs to address system memory.



Pin Functions

LABEL	PIN	DESCRIPTION
V_{CC}	40	+5 Volt Power Supply
GND	20	Ground
D0–D7	28–21	Data bus *(Input/Output – Active High – 3-State)*
A0–A9	1–10	Address bus (main) *(Input/Output – Active High – 3-State)*
A10–A15	11–16	Address bus (extension) *(Output – Active High – 3-State)*
$\overline{CE}$	30	Chip Enable *(Input – Active Low)*
R/$\overline{W}$	31	Read/Write mode select *(Input – Read-High/Write-Low)*
$\overline{MREQ}$	18	Memory REQuest *(Output – Active Low)*
$\overline{BREQ}$	37	Bus REQuest *(Output – Active Low)*
$\overline{BACK}$	38	Bus ACKnowledge *(Input – Active Low)*
CLK	19	Clock *(Input)*
$\overline{RESET}$	32	Master Reset *(Input – Active Low)*
$\overline{WAIT}$	17	Wait *(Input – Active Low)*

4.2 PIN FUNCTIONS

D0–D7 constitute an 8–bit bidirectional data bus. The bus is configured as an *output* during read accesses to the PF474. At all other times it is configured as an *input*.

A0–A9 constitute a 10–bit bidirectional address bus. The bus is configured as an *output* during PF474 DMA operation. At all other times it is configured as an *input*.

A10–A15 constitute a 6–bit 3–state address bus. The bus is configured as an *output* during PF474 DMA operation. At all other times it is in the 3–state *off* condition. These six outputs together with the ten outputs A0–A9 enable the PF474 to generate a 16–bit address during DMA operation to facilitate simple interface to most microprocessor systems.

$\overline{\text{CE}}$ must be asserted (low) to read from or write to the PF474's internal memory. It must not be asserted during DMA. The address bus must be valid at the falling edge and, during write cycles, the data bus must be valid at the rising edge.

R/$\overline{\text{W}}$ is an input which selects read or write mode during non–DMA accesses to the PF474 internal memory.

$\overline{\text{MREQ}}$, when asserted (low), indicates that the PF474 is requesting a memory access, and that A0–A15 contain a valid address. $\overline{\text{MREQ}}$ is only active during DMA operation.

$\overline{\text{BREQ}}$ is used to request control of the data and address buses in preparation for a PF474 DMA operation.

$\overline{\text{BACK}}$ is asserted to the PF474 in response to the assertion of $\overline{\text{BREQ}}$ by the PF474. This signal informs the PF474 that it has control of the address and data lines. $\overline{\text{BACK}}$ must then remain asserted until the PF474 is no longer asserting $\overline{\text{BREQ}}$. This input is ignored when $\overline{\text{BREQ}}$ is high.

CLK is a single phase TTL square wave clock.

$\overline{\text{RESET}}$ master clears the PF474. While $\overline{\text{RESET}}$ is asserted (low), D0–D7 and A0–A9 are configured as inputs, A10–A15 are held in their high impedance state, and $\overline{\text{MREQ}}$ and $\overline{\text{BREQ}}$ are held high. The reset function terminates any PF474 operation in progress including DMA transfers but does not alter previously entered data. Note, however, that the Ranked List will be temporarily scrambled until a new Proximity Computation has been performed. $\overline{\text{RESET}}$ must be asserted for a minimum of 3 1/2 clock periods.

$\overline{\text{WAIT}}$ is used to lengthen DMA cycles, permitting the use of arbitrarily slow external memories. $\overline{\text{WAIT}}$ is sampled and latched at each rising edge of the clock.

4.3 RATINGS AND CHARACTERISTICS

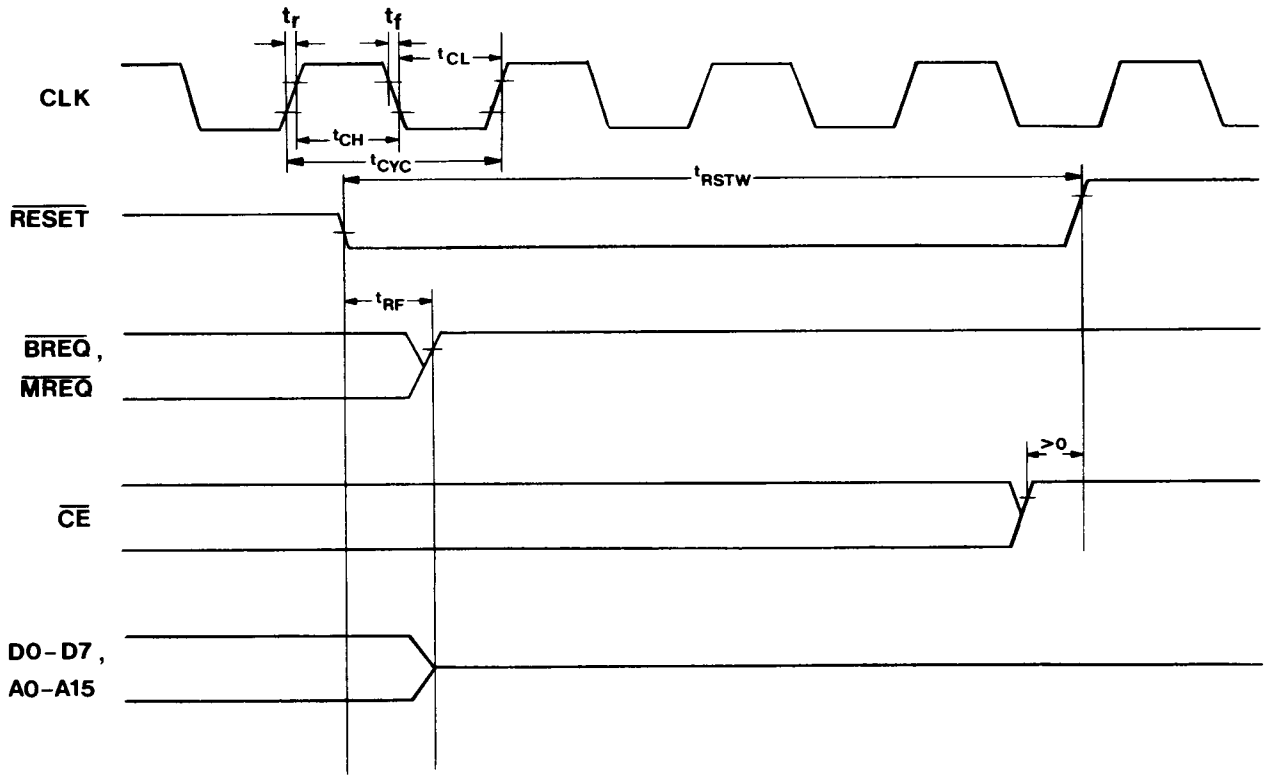
Absolute Maximum Ratings				
Symbol	Parameter	Limits		Units
		Min	Max	
V_{CC}	Supply Voltage	–0.3	7.0	V
V_{IN}	Input Voltage	–0.3	7.0	V
T_A	Operating Temperature	0	70	Degrees °C
T_{STG}	Storage Temperature	–55	125	Degrees °C

D.C. Characteristics					
Symbol	Description	Limits			Units
		Min	Typ	Max	
V_{CC}	Power Supply Voltage	4.75	5.0	5.25	V
I_{CC}	Power Supply Current		300		mA
V_{IH}	Input HIGH Voltage	2.0		$V_{CC}+0.5$	V
V_{IL}	Input LOW Voltage	–0.3		0.8	V
I_{LI}	Input Current			20	μA
V_{OH}	Output HIGH Voltage (V_{CC}=min, I_{OH}=–400 μA)	2.8	3.6		V
V_{OH}	Output LOW Voltage (V_{CC}=min, I_{OL}=1.6 mA)		0.25	0.4	V
I_{LO}	Output Leakage Current			20	μA

4.4 INTERFACE TIMING

Although the PF474 performs internal operations beyond the power of most sophisticated computers, the interface requirements are scarcely more complex than those of a simple memory circuit. The basic I/O operations and the DMA protocol are designed for easy interface and high throughput. Below are waveform diagrams summarizing the interface and timing requirements of the PF474.

4.4.1 Reset.



This diagram shows the use of $\overline{\text{RESET}}$. Asserting $\overline{\text{RESET}}$ terminates all PF474 operations, including DMA. The $\overline{\text{RESET}}$ signal does not affect PF474 memory, such as parameter tables, option settings, and String Memories. $\overline{\text{RESET}}$ must be continuously asserted for at least three and one-half clock periods, asynchronous to clock CLK. $\overline{\text{RESET}}$ forces $\overline{\text{MREQ}}$ and $\overline{\text{BREQ}}$ high (not asserted), configures data and address lines (0–9) as inputs, and places address lines (10–15) in the high impedance *off* state. $\overline{\text{RESET}}$ may be asserted independent of the state of $\overline{\text{CE}}$ and $\overline{\text{BACK}}$, but $\overline{\text{CE}}$ must be high at the end of the $\overline{\text{RESET}}$ cycle.

4.4.2 Read/Write Cycles.

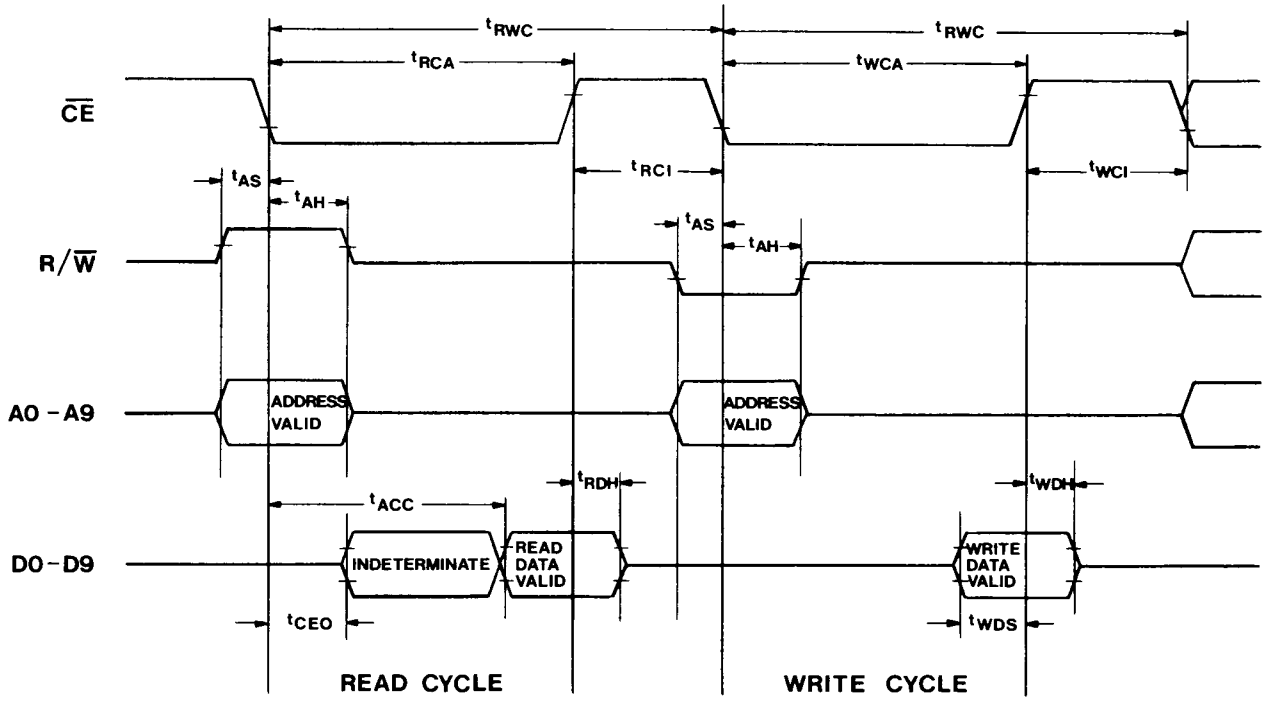
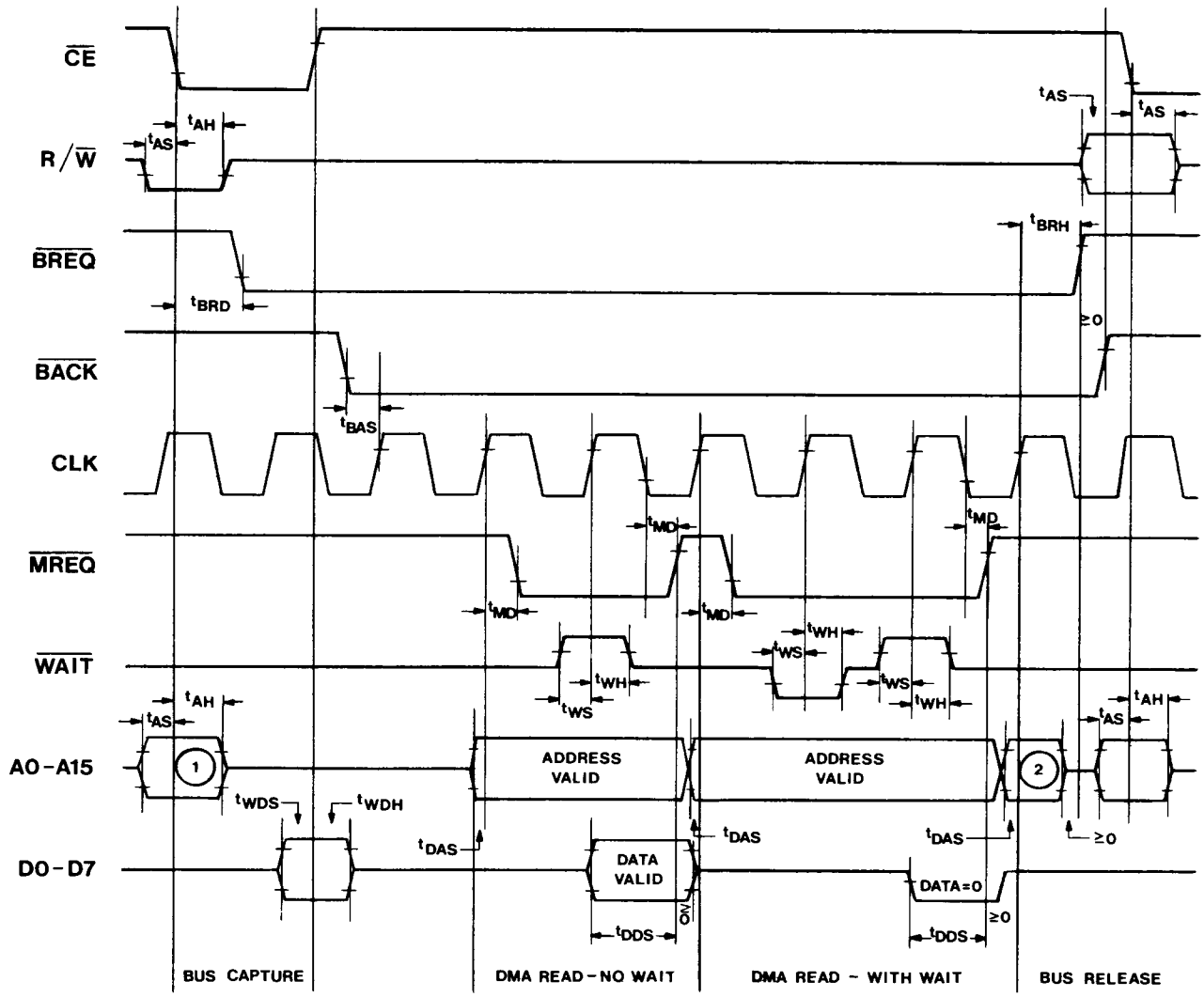


Figure 3. Read/Write Cycle Timing

This diagram shows the signals involved in read and write operations. The address and R/$\overline{W}$ signals must be valid t_{AS} before the falling edge of $\overline{CE}$. These lines are sampled at the falling edge of $\overline{CE}$ and need to be held valid only t_{AH} after $\overline{CE}$ is asserted. In the read cycle (shown to the left), $\overline{CE}$ must remain low until the data from the PF474 becomes valid and has been sampled by the system. In the write cycle, the data is sampled by the PF474 at the rising edge of $\overline{CE}$.

4.4.3 DMA Cycle.



1) Write of the second DMA Register
2) Next sequential address

Figure 4. DMA Cycle Timing

This diagram displays the features of PF474 DMA operation. It consists of four sections: bus capture, a DMA operation without WAIT states (such as might be used to read a fairly fast memory), a DMA operation with a single WAIT state inserted (for reading a moderately slow memory), and bus release.

DMA operation is invoked by writing the address of the source string into the PF474 DMA Register. The left edge of the diagram shows the second address byte being written into the DMA address register. Note that the PF474 requests the bus by dropping $\overline{\text{BREQ}}$ during the write operation, while $\overline{\text{CE}}$ is still low (even if String Control is still busy). $\overline{\text{BACK}}$ signals to the PF474 that the CPU has relinquished

control of the system bus so that DMA operation may begin. $\overline{BACK}$ may be delayed indefinitely without ill effect, but must stay asserted throughout DMA. Premature release will cause unpredictable results. The PF474 will drive the address lines as soon as $\overline{BACK}$ is asserted (low), and CE4 is released (high). $\overline{MREQ}$ falls at least one clock cycle later. If the Proximity Computer is still busy with a previous computation, $\overline{MREQ}$ will wait until the previous Proximity Computation has been completed.

The next section of the diagram shows one DMA cycle without the insertion of WAIT states. When performing DMA read operations, the PF474 generates $\overline{MREQ}$ to request an external memory read operation. Address lines A0–A15 are valid before $\overline{MREQ}$ falls and remain valid while $\overline{MREQ}$ is low. $\overline{MREQ}$ operates synchronous to the clock: $\overline{MREQ}$ falls after the rising edge of the clock, and rises after the second falling clock edge. The Data lines D0–D7 are sampled at the rising edge of $\overline{MREQ}$. Therefore, for maximum DMA throughput, the memory must be able to respond with valid data one clock cycle after $\overline{MREQ}$ falls.

For slower memories, WAIT states should be inserted via the $\overline{WAIT}$ signal as illustrated in the third part of the diagram. When the memory accessed has a slower response than one clock cycle, each $\overline{MREQ}$ cycle time may be adjusted to any number of clock cycles beyond the two required for normal operation through the use of $\overline{WAIT}$. $\overline{WAIT}$ is sampled at each rising edge of the clock pulse while $\overline{MREQ}$ is low. When $\overline{WAIT}$ is sampled low, the DMA read operation is extended. One clock pulse (1/2 clock cycle) after $\overline{WAIT}$ is sampled high, $\overline{MREQ}$ will rise, and the data will be sampled, completing the DMA cycle.

The right edge of the diagram shows the release of the bus by the PF474. This will occur after the PF474 has read a NULL character. $\overline{MREQ}$ remains high after the terminating NULL, although the next address is temporarily placed on the bus. $\overline{BREQ}$, which remained low for the duration of DMA, rises, signifying the relinquishing of bus control. The CPU may resume control immediately.

A.C. Characteristics

Cycle	Symbol	Parameter	Limits		Units
			Min	Max	
Clock	f	Frequency of Operation	1	– [1]	MHz
	t_{CYC}	Clock Period *(cycle time)*	– [2]	1000	nsec
	t_{CL}	Clock Low	– [3]		nsec
	t_{CH}	Clock High	– [4]		nsec
	t_r	Clock Rise Time		20	nsec
	t_r	Clock Fall Time		20	nsec
Reset	t_{RSTW}	$\overline{\text{RESET}}$ Pulse Width	3.5		t_{CYC}
	t_{RF}	$\overline{\text{RESET}}$ to Bus Release		100	nsec
Read/ Write	t_{RWC}	Read/Write Cycle Time	400		nsec
	t_{AS}	Address Setup	10		nsec
	t_{AH}	Address Hold	40		nsec
Read	t_{RCA}	Read Cycle Active	250	1500	nsec
	t_{RCI}	Read Cycle Inactive	150		nsec
	t_{ACC}	Read Access		250	nsec
	t_{CEO}	Chip Enable to Output Delay	40		nsec
	t_{RDH}	Read Data Hold	40		nsec
Write	t_{WCA}	Write Cycle Active	200	1500	nsec
	t_{WCI}	Write Cycle Inactive	200		nsec
	t_{WDS}	Write Data Setup	20		nsec
	t_{WDH}	Write Data Hold	50		nsec
DMA	t_{DMA}	DMA Cycle Time	2		t_{CYC}
	t_{BRD}	Bus Request Delay from $\overline{\text{CE}}$		125	nsec
	t_{BAS}	Bus Acknowledge Setup	0		nsec
	t_{MD}	Memory Request Delay		60	nsec
	t_{DAS}	DMA Address Setup	20		nsec
	t_{WS}	Wait Setup	20		nsec
	t_{WH}	Wait Hold	50		nsec
	t_{DDS}	DMA Data Setup	30		nsec
	t_{BRH}	Bus Request Hold		80	nsec

1. f_{max} = 2.5 for PF474–25, 3.0 for PF474–30, 3.6 for PF474–3.6, and 4.0 for PF474–40.
2. $1000/f_{(max)}$
3. $t_{CYC}(min)/2 - 5$
4. $t_{CYC}(min)/2 - 5$